Multi-Value-Functions: Efficient Automatic Action Hierarchies for Multiple Goal MDPs

Andrew W. Moore
Carnegie Mellon University
awm@cs.cmu.edu

Leemon C. Baird
Carnegie Mellon University
leemon@cs.cmu.edu

Leslie Kaelbling
Brown University
lpk@cs.brown.edu

Abstract

If you have planned to achieve one particular goal in a stochastic delayed rewards problem and then someone asks about a different goal what should you do? What if you need to be ready to quickly supply an answer for any possible goal? This paper shows that by using a new kind of automatically generated abstract action hierarchy that with N states, preparing for all of N possible goals can be much much cheaper than N times the work of preparing for one goal. In goal-based Markov Decision Problems, it is usual to generate a policy $\pi(x)$, mapping states to actions, and a value function $J(x)$, mapping states to an estimate of minimum expected cost-to-goal, starting at x. In this paper we will use the terminology that a **multi-policy** $\pi^\star(x,y)$ (for all state-pairs (x,y)) maps a state x to the first action it should take in order to reach y with expected minimum cost and a **multi-value-function** $J^\star(x,y)$ is a definition of this minimum cost. Building these objects quickly and with little memory is the main purpose of this paper, but a secondary result is a natural, automatic, way to create a set of parsimonious yet powerful abstract actions for MDPs. The paper concludes with a set of empirical results on increasingly large MDPs.

1 Introduction

In goal-based Markov Decision Problems, it is usual to generate a policy and a value function for a single goal. In 1993, Kaelbling introduced the HDG (Hierarchical Distance to Goal) algorithm [Kaelbling, 1993], which considered the more general formalism of arbitrary goal states as well as start states. Given N states, HDG considered the question of how to generate and execute an approximation to the multi-policy without requiring $O(N^2)$ memory for the data structures.

In this paper we introduce new data structures and algorithms that exploit this insight while automatically choosing where to place *landmarks* (points to aim for), and choosing which states belong to which landmark, without requiring pre-defined distance metrics (such as the Euclidean heuristic of the original paper) or predefined partitions. Instead, the measure of closeness used is the true minimal expected cost measure. Generating these locations, partitions, and policies from scratch involves some interesting "chicken and egg" problems, especially if we need to exploit the hierarchy whilst building it (necessary if we are to save not only memory but also computation). We will describe how this can be addressed with a new branch-and-bound procedure in combination with prioritized sweeping [Moore and Atkeson, 1993], an algorithm for improving value functions in the face of small alterations in an MDP structure.

Another motivation is to help answer the two questions, "Where should Abstract Actions come from?". Is it necessary to have some high-level prior understanding of the class of tasks at hand in order to decide which abstractions are beneficial? This may be of use to recent algorithms that attempt to exploit abstract actions but require them to be predefined (such as [Precup and Sutton, 1998; Singh, 1991; Dayan and Hinton, 1993; Dietterich, 1998] and HDG itself).

2 The Airport Hierarchy

This algorithm takes, as input, an MDP which is specified by a set of N states, and for each state a set of actions, and for each state-action pair, a set of *outcomes* along with the outcome probabilities. The $OUTCOMES()$ set of a state-action-pair (x,a) is the set of states $\{z\}$ such that $P($ next state $= z \mid$ this state $= x$, this action $= a) > 0$. We make the strong assumption that the MDP is sparse: for all states the number of actions is $<< N$ and for all state-action pairs, the number of outcomes is $<< N$. This assumption is very frequently true in navigation, dynamics and inventory MDPs. Furthermore, recent work [Kearns *et al.*, 1998] indicates that even non-sparse MDPs may usually be transformable to sparse MDPs with little loss in optimality. In addition to the state transition probabilities, the algorithm also inputs the non-negative one-step cost function $C(x,a)$ specifying the penalty we pay for applying a in state x. Note that conventional dynamic programming theory tells us that the optimal policy and multi value function must obey $J^\star(x,y) = min_a Q^\star(x,y,a)$ and $\pi^\star(x,y) = argmin_a Q^\star(x,y,a)$ where

$$Q^\star(x,y,a) = C(x,a) + \sum_{z \in \mathrm{OUTCOMES}(x,a)} P(z \mid x,a) J^\star(z,y)$$

The new structure is called the Airport Hierarchy. It differs from HDG and is more similar to Feudal Learning [Dayan and Hinton, 1993] in that each state is a member of many partitions: some large and abstract, others small and specific. It differs from Feudal Learning and other partitioning structures (such as G-learning [Chapman and Kaelbling, 1991] and PartiGame [Moore, 1994]) in that junior partitions are not necessarily subsets of their seniors, and many partitions are overlapping. We will begin by defining the properties that we *do* require from these partitions, before proceeding to execution and generation algorithms that exploit these properties. Proofs will be included in a later version of this paper.

2.1 Formalities

We define, for a given state y that we have decided to turn into an airport:

$$\mathit{INS}(y) = \{\text{states that know how to get to } y\} \qquad (1)$$

Any member of $\mathit{INS}(y)$ will have, cached away in the airports structure, knowledge of $\pi^\star(x,y)$, the truly optimal first action to take in getting to y and $J^\star(x,y)$, the true minimum expected cost of getting to y. Define

$$\mathit{Level}(y) = \text{The "Level" of airport } y \qquad (2)$$

Where $\mathit{Level}(y) = 0$ means that y is a maximally senior airport: the larger the level, the lower the seniority. As part of the construction we will insist that

$$|\ \mathit{INS}(x)\ | \geq N/2^{\mathit{Level}(x)} \qquad (3)$$

Thus, if y is a level i airport then there are at least $N/2^i$ states that know the optimal value and policy for reaching y. We also insist that

$$x \in \mathit{INS}(y) \wedge z \notin \mathit{INS}(y) \Rightarrow J^\star(x,y) \leq J^\star(z,y) \qquad (4)$$

Thus, the set of n states that knows how to get to y are the n states with minimal expected cost to get to y. Our final two requirements are

- If $\mathit{Level}(y) \geq 1$ then $\mathit{INS}(y)$ includes at least k senior airports (x is senior to z if and only if $\mathit{Level}(x) < \mathit{Level}(z)$).
- There are k airports at level 0, $2k$ at level 1, $4k$ at level 2, etc, with the number of airports doubling at each level until we run out of states.

Because the number of airports doubles at each level, the maximum level rises logarithmically with N. The total amount of memory needed to represent all knowledge about all level i airports is thus, in the best case, $O(N/2^i)$ per airport times $k2^i$ level i airports, which is thus $O(kN)$, and so the total amount of memory needed over all levels is $O(kN \log N)$ in the best case.

Example: Figures 1 and 2 depict aspects of the airport hierarchy for a simple grid-world.

2.2 Choosing Actions

How is $\pi^\star(x,y)$ approximated by the airport hierarchy? Call the approximation $\hat{\pi}(x,y)$. To choose the first action to take if we hope eventually to reach y cheaply, we use the HDG insight. If x is far enough from y that it

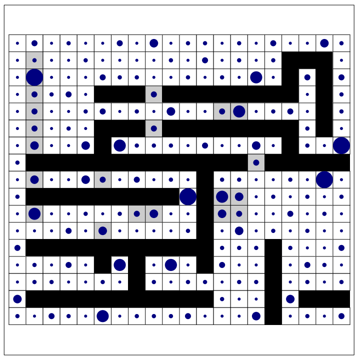
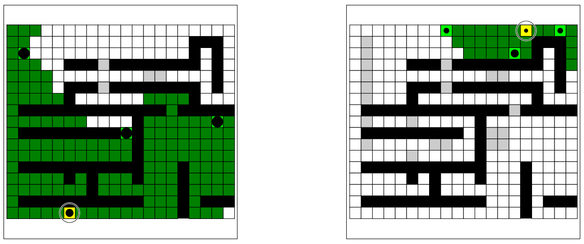


Figure 1: A simple maze constructed with $k = 4$ top-level airports. The light gray areas are cells in which transitions are completely random (to a random neighbor). In white cells, there is a 0.9 probability that we'll move in the direction (N, S, E or W) we request, and a $p_{\mathrm{rand}} = 0.1$ chance we'll move to a random neighbor. The disks denote the airports: the larger the disk, the more senior the airport.



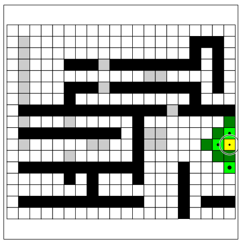


Figure 2: Each figure shows an airport y (circled) and $\mathit{INS}(y)$ (dark grey). The airport levels are 1, 3 and 5 respectively. The $k = 3$ senior airports that can reach the targets are also shown.

doesn't know how to get to y directly (i.e., if $x \notin \mathit{INS}(y)$), we pick some intermediate target that we *do* know how to get to, and from which we will be able to make progress towards y. The only remaining question is how to choose a good target from the airport hierarchy.

To do this, we must define one extra object, called $\mathit{SINS}(y)$, a special set of states that know how to get to y by travelling through increasingly less senior airports. Define

$$\mathit{SINS}^0(y) = \{y\}$$
$$x \in \mathit{SINS}^{k+1}(y) \Leftrightarrow x \notin \mathit{SINS}^k(y) \wedge$$
$$\exists z\ .\ (z \in \mathit{SINS}^k(y) \wedge x \in \mathit{INS}(z) \wedge \mathit{Level}(x) > \mathit{Level}(z))$$

So that, for example, $\mathit{SINS}^1(y)$ consists of those states in $\mathit{INS}(y)$ that happen to have a more senior airport level to $\mathit{Level}(y)$. Then write $\mathit{SINS}(y) = \bigcup_k \mathit{SINS}^k(y)$. Now we can define the target that x first aims for. We will deal in turn with three cases: states that immediately know how to get to y; states that are in $\mathit{SINS}(y)$; and all other states.

- If $x \in \mathit{INS}(y)$ then define $\hat{\pi}(x,y) = \pi^\star(x,y)$, and define $\hat{J}(x,y) = J^\star(x,y)$. These values can be read, in constant time, from the information cached in the airport hierarchy.

- If $x \notin \mathit{INS}(y)$, but $x \in \mathit{SINS}(y)$ then assume that x was entered into $\mathit{SINS}(y)$ at level $k+1$, so that $x \in \mathit{SINS}^{k+1}(y)$ but $x \notin \mathit{SINS}^{k}(y)$. Then define
$$\hat{\pi}(x,y) = \mathit{argmin}_{z \in \mathit{SINS}^{k}(y)} J^{\star}(x,z) + \hat{J}(z,y)$$
$$\hat{J}(x,y) = \mathit{min}_{z \in \mathit{SINS}^{k}(y)} J^{\star}(x,z) + \hat{J}(z,y)$$

- Finally, if $x \notin \mathit{INS}(y)$ and $x \notin \mathit{SINS}(y)$ then define $\mathit{Common}(x, \mathit{SINS}(y))$ to be the set of states $\{z\}$ such that $x \in \mathit{INS}(z) \wedge z \in \mathit{SINS}(y)$. And then define
$$\hat{\pi}(x,y) = \mathit{argmin}_{z \in \mathrm{Common}(x,\mathrm{SINS}(y))} J^{\star}(x,z) + \hat{J}(z,y)$$
$$\hat{J}(x,y) = \mathit{min}_{z \in \mathrm{Common}(x,\mathrm{SINS}(y))} J^{\star}(x,z) + \hat{J}(z,y)$$

$\mathit{Common}(x, \mathit{SINS}(y))$ cannot be empty because $\mathit{SINS}(y)$ must contain a Level 0 airport.

Intuitively, the process of creating $\mathit{SINS}(y)$ can be seen as dynamically creating a goal-state-dependent set of landmarks (using HDG terminology) for getting to y. This set of landmarks is useful specifically for y because it has a higher density of landmarks closer to y (with closeness defined in terms of minimal expected cost). By construction (easy to prove) this set is small: $O(k \log_2 N)$.

Figure 3 shows the set of airports considered during one such decision. Although the policies resulting from this scheme may be suboptimal, we can prove that they will always reach the goal in finite time, and empirically the regret (mean loss compared with truly optimal behavior) is very small.

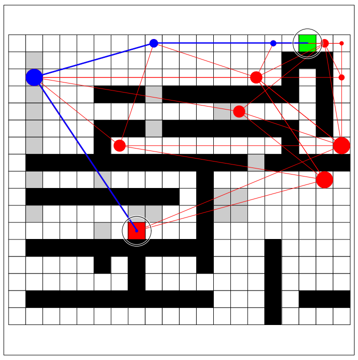


Figure 3: Planning the next action from the centrally circled state. The start state (x) is the cell surrounded by the lower large circle. The goal state (y) is surrounded by the upper large circle. The set $\mathit{SINS}(y)$ are depicted by small discs, with radii proportional to seniority.

2.3 Efficiently building the airport hierarchy

We say a state is *unassigned* if it is not an airport at any level in the hierarchy. The airport construction algorithm begins with all states unassigned, and ends with no states unassigned. Whenever an airport y becomes assigned, $\mathit{Level}(y)$ is chosen, the set $\mathit{INS}(y)$ is chosen, and for all $x \in \mathit{INS}(y)$, $J^{\star}(x,y)$ and $\pi^{\star}(x,y)$ are computed. All this information is cached permanently away in the airport hierarchy for future use during the remaining airport construction and policy execution.

We will now discuss the **AddAirport** operation, which picks one state and assigns it.

2.4 Step One: Choose y

We defer discussion of this step until Section 2.7.

2.5 Step Two: Choose y's Level

This step is trivial. y is simply given the most senior airport level available, subject to the constraint that there are no more than $k2^L$ airports at level L. Thus, if so far, m airports have been assigned, y's level is $\log_2(m/k)$.

2.6 Step Three: Construct $\mathit{INS}(y)$

To understand the job we must do here, we will begin by considering two simplified scenarios: an inefficient method for non-deterministic MDPs and an efficient method for deterministic MDPs. Finally, we'll tackle the efficient non-deterministic case.

Constructing $\mathit{INS}(y)$ inefficiently

We would simply take the following steps:

1. Run an MDP solver such as value iteration, policy iteration, or modified policy iteration [Bertsekas and Tsitsiklis, 1989] to compute $\pi^{\star}(x,y)$ and $J^{\star}(x,y)$ for all states x in the MDP.
2. Define the set $\mathit{Bests}^{T}(y) = \{x$ such that $J^{\star}(x,y)$ is among the T lowest values. $\}$, with ties broken arbitrarily to ensure T elements in the set. Define $T^{\star}$ to be the minimum T such that $T \geq N/2^{\mathit{Level}(y)}$ and $\mathit{Bests}^{T}(y)$ contains at least k senior airports. Then simply select $\mathit{INS}(y) = \mathit{Bests}^{T^{\star}}(y)$ and store the $J^{\star}(x,y)$ and $\pi^{\star}(x,y)$ values in the airport hierarchy only for those xs for which $x \in \mathit{INS}(y)$.

The objection to this approach is that it is too expensive: after having computed the $\mathit{INS}(y)$ set for every y in the MDP, we will have needed to perform N full dynamic programs, meaning $O(N^2 k_{\mathrm{crit}})$ operations in total to build the hierarchy, where k_{crit} is the average number of value iterations needed for dynamic programming convergence.

Constructing $\mathit{INS}(y)$ for deterministic systems

This too would be simple. We can perform uniform-cost breadth-first search back from y until the set of states added during the search become sufficiently large and contains the required number of senior airports. Now the work per airport would be small for junior airports with small $\mathit{INS}(y)$ state sets, resulting in an algorithm that in the best case could be as cheap as $O(N \log N)$.

The efficient non-deterministic case

Again, we will work backwards from y, incrementally adding states. The problem is that for a non-deterministic system there is always a probability that the system being controlled will leave the set of states added so far. How, then, can we compute the optimal value function that takes into account the possibility of such exits?

Let us briefly take a break from the larger question of computing $\mathit{INS}(y)$, and instead focus on a simpler question: given an incomplete portion of a goal-based MDP, how can we approximate $J^\star(x,y)$ over the states in this incomplete portion (which we will call S)? Figure 4 gives an example of such a problem, with five states (including the goal state, Y) constituting S, and several outcomes that lead to states outside this set marked with **?** symbols. We will find an optimistic lower bound, $J^{\rm opt}(x,y)$, and a pessimistic upper bound, $J^{\rm pess}(x,y)$, on $J^\star(x,y)$ so that

$$\forall x \in S \ . \ J^{\rm opt}(x,y) \le J^\star(x,y) \le J^{\rm pess}(x,y) \qquad (5)$$

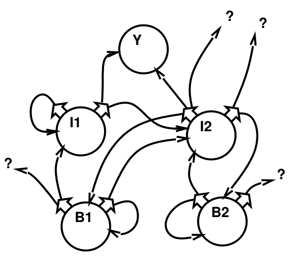


Figure 4: An example of a sub-MDP in the process of construction from five states $S = \{Y, I_1, I_2, B_1, B_2\}$. The thick polygonal arrows denote actions. The thin arrows denote the possible stochastic outcomes if the action is taken. Assume that all outcomes cost 1 unit, and for a given state-action pair, all depicted outcomes are equiprobable.

Constructing $J^{\rm opt}(x,y)$

We will construct a new, small, MDP made up of the states in S, plus one new state X. This construction will be guaranteed not to overestimate $J^\star(x,y)$.

Before proceeding, we will partition S into two subsets of states. Define

$$\begin{aligned} \mathit{Internals} &= \{x \in S \text{ such that } \forall z \in \mathit{Preds}(x) \ . \ z \in S\} \\ \mathit{Borders} &= \{x \in S \text{ such that } \exists z \ . \ z \in \mathit{Preds}(x) \wedge z \notin S\} \end{aligned}$$

where $z \in \mathit{Preds}(x)$ means "z is an immediate predecessor of x", so that $z \in \mathit{Preds}(x) \Leftrightarrow \exists a$ such that $P($ next state $= x \mid$ this state $= z$ and action $= a) > 0$.

Border states have some predecessor *outside* S, while internal states are ones in which all predecessors are inside S. In Figure 4, assume that Y, I_1, and I_2 are all internal states, and B_1 and B_2 are border states.

For each of the **?** symbols in Figure 4, what is the best thing that could happen? The first thought is that they might head straight to a state outside S, paying their one-step outcome cost, and then this state might be able to jump to the goal with zero cost. But this is not possible. If the goal is an internal state, then no state outside S can jump directly to the goal, nor indeed to any other internal state. So the best thing that could happen is that they jump to some state outside S and then with zero cost jump to the most favorable of the border states. We model this by adding a new, fake, state called X to the system, which can with zero cost jump to any of the border states that it chooses. The **?** symbols all jump to X. Figure 5 shows this construction for our example. Having added this extra state, we run an MDP solver to compute its value function, which we define to be $J^{\rm opt}(x,y)$.

Constructing $J^{\rm pess}(x,y)$

We must now ask "what is the worst thing that can

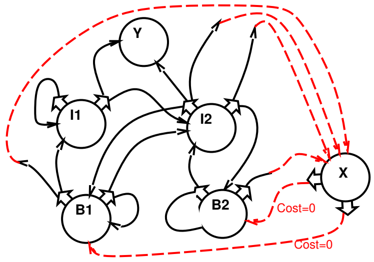


Figure 5: The optimistic interpretation of the subMDP of Figure 4.

happen at any of the **?** symbols?". Unfortunately, it is arbitrarily bad. Even if there were an upper bound on a single transition cost in the MDP, hard-to-escape loops (caused by a probability-of-transition-to-self very close to 1) could create arbitrarily expensive expected costs to the goal.

The good news is that if $\mathit{Level}(y) > 0$, we may be able to take advantage of information cached in the partially constructed airport hierarchy. This requires that at least one state in S have already been added to the hierarchy as an airport, and that one or more other states in S know how to get to this airport.

If there exists $w \in S$ and $x \in S$ such that w is an airport and $x \in \mathit{INS}(w)$ then we will add an extra action to x, with cost $J^\star(x,w)$ (which we can look up in the hierarchy) and with next state w with probability 1. Adding this action simply expresses the already established fact that we know we can get to w from x by some (not necessarily explicitly known) means, and the expected cost of doing so is $J^\star(x,w)$. Expressing this fact cannot cause the system to underestimate the cost of travelling from x to y. Adding these new fake actions can have a dramatic effect on the worst case cost estimates. In Figure 4, the worst case value for all states other than Y is infinite. But suppose we add the extra knowledge that we already know that travelling from I_2 to I_1 costs, say, an expected 3 units (see Figure 6). Then the optimal policy in the supplemented MDP will have costs $J^{\rm pess}(I_1,Y) = 5$, $J^{\rm pess}(I_2,Y) = 8$, $J^{\rm pess}(B_1,Y) = 10$ and $J^{\rm pess}(B_2,Y) = 10$.

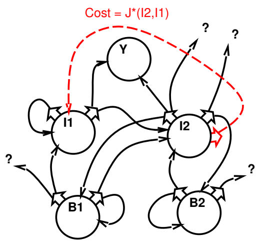


Figure 6: The pessimistic interpretation of the subMDP of Figure 4, assuming that $I_2 \in \mathit{INS}(I_1)$.

Constructing $\mathit{INS}(y)$

We now return to the question of constructing $\mathit{INS}(y)$. The algorithm is:

1. Let $S = \{y\}$, $\mathit{Borders} = \{y\}$, $\mathit{Internals} = \{\}$.
2. Compute $J^{\rm opt}(x,y)$ and $J^{\rm pess}(x,y)$ for all $x \in S$ using the modified versions of S described above.

3. Define the set $Bests^T(y) = \{x$ such that $J^{\rm opt}(x,y)$ is among the T lowest values $\}$. Define $T^\star$ to be the minimum T, if it exists, such that $T \geq N/2^{Level(y)}$ and $Bests^T(y)$ contains at least k senior airports.

   If such a $T^\star$ exists, check the tightness on our lower and upper bounds on $J^\star(x,y)$ for all states in $Bests^{T^\star}(y)$. If

$$\forall x \in Bests^{T^\star}(y) \ . \ J^{\rm pess}(x,y) - J^{\rm opt}(x,y) < \epsilon \quad (6)$$

   then halt, adding y into the airport hierarchy, along with its chosen level, and $INS(y) = Bests^{T^\star}(y)$, and remembering for future use all the $J^\star(x,y)$ values for $x \in INS(y)$, approximating $J^\star(x,y)$ by $(J^{\rm opt}(x,y) + J^{\rm pess}(x,y))/2$.

4. If we did not halt, grow S. Pick the border state with minimum $J^{\rm opt}(x,y)$, add all its predecessors into S and *Borders*. Check which states have all their immediate predecessors in S and promote them into *Internals*. Goto 2.

Remarks:
Each time $J^{\rm opt}(x,y)$ and $J^{\rm pess}(x,y)$ are recomputed, few of the states in S change their values significantly, except for newly added states and their neighbors. To take advantage of this, we use prioritized sweeping [Moore and Atkeson, 1993]. This was used in the experiments reported in Section 3, and without it, the computational costs were empirically 5 to 100 times slower.

In order to access the immediate predecessors in the MDP, we must create a backwards model, which is an array which, for each state x, maps x to *Preds*(x). This can be built once, with a construction cost linear in the number of states.

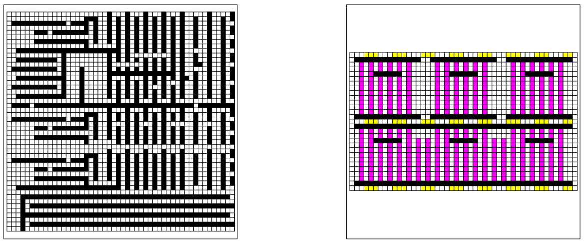
Why should we expect that the upper and lower bounds on the value function will be tight? Not all domains will provide tightness: for example, expander graphs are likely to be problematic. But domains with a notion of *locality* are likely to achieve tight bounds on states close to the goal quite early on.

2.7 Choosing airport locations

When the hierarchy is being built, level 0 airports are assigned first, then level 1, etc. The location of a new airport is always chosen to be the state furthest from any previous airport (in terms of expected cost). Because of the construction of the hierarchy, this can always be found efficiently. Notice that no prior assumptions (such as Euclidean distance) are needed for airport determination: it is fully automatic.

3 Results

Table 1 presents results on various stochastic gridworlds and an inventory management problem. The gridworlds all obey the same rules as our initial example. Problems Expand1 through Expand15 are gridworlds of increasing size, produced by gluing together a short fat maze on top of itself. This allows us to see how costs increase as the maze size increases linearly. OneWay (middle of Figure 7) is a maze with one-way walls, and serves to demonstrate that it is not necessary for actions to be reversible for airports to be applicable. The table compares the memory needed if a full $\pi(x,y)$ table were used to that needed by the airport hierarchy. It compares the time needed to build the airport hierarchy conventionally with the time needed using our algorithm. And it shows the mean regret (loss due to using our approximate policy, $\hat{\pi}(x,y)$, instead of the optimal policy, $\pi^\star(x,y)$). To give the regret some context it also shows the mean path cost. Figure 8 graphically depicts the tradeoff as the number of states is increased. Further empirical results, for which there is no room here, show that the algorithm work over a wide range of stochasticity, that there is not strong sensitivity to the ϵ threshold, and that clever placement of the airports is beneficial.



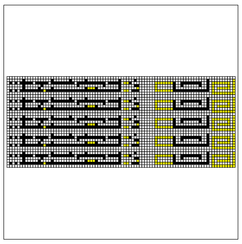


Figure 7: TopLeft: The Medium Maze. TopRight: The OneWayMaze (state transitions are permitted rightwards through the grey vertical bars, but not leftwards). BottomLeft: The ExpanderMaze of size five.

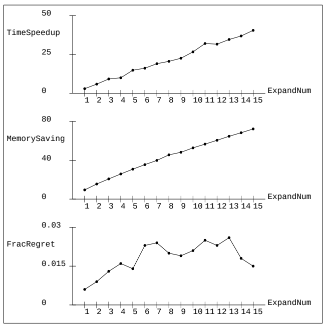


Figure 8: The relative performance of using versus not using airports for different sizes of the Expand maze of Figure 7. Top graph: number of times by which the airports are faster than non-airports in computing $J^\star(x,y)$ and $\pi^\star(x,y)$. Middle graph: the number of times more memory that non-airports need. Bottom graph: the average regret of the airports policy, expressed as a fraction of path length.

4 Conclusion

This paper has been about efficiently computing and caching a hierarchy that allows us to approximate $J^\star(x,y)$ and $\pi^\star(x,y)$ for all pair of states (x,y). When is

Name	Num States	Slow Memory (Words)	Airports Memory (Words)	Memory Saving Factor	Slow Time (secs)	Airports Time (secs)	Speed -up Factor	Mean $J^\star(x, y)$ cost	Airports Mean Regret	Fraction Regret
Small	246	0.061	0.011	5.4	18	7	2.6	30	0.28	0.009
Medium	1477	2.18	0.055	39.3	3064	216	14.2	39	0.46	0.012
Big	6480	41.990	0.700	60.0	69822	1429	48.9	104	0.51	0.005
OneWay	1180	1.392	0.085	16.4	2301	219	10.5	99	2.84	0.029
Inventory	5313	28.228	2.507	11.3	55653	7518	7.4	62	6.90	0.111
Expand1	485	0.235	0.025	9.5	97	33	2.9	39	0.25	0.006
Expand2	970	0.941	0.061	15.5	557	95	5.9	41	0.38	0.009
Expand3	1455	2.117	0.102	20.8	1600	174	9.2	44	0.56	0.013
Expand4	1940	3.764	0.145	25.9	2570	258	10.0	49	0.76	0.016
Expand5	2425	5.881	0.191	30.8	4910	331	14.8	52	0.74	0.014
Expand6	2910	8.468	0.239	35.5	6838	424	16.1	55	1.27	0.023
Expand7	3395	11.526	0.289	39.9	10185	535	19.0	58	1.38	0.024
Expand8	3880	15.054	0.330	45.6	12940	630	20.5	60	1.18	0.020
Expand9	4365	19.053	0.395	48.3	16914	750	22.6	62	1.17	0.019
Expand10	4850	23.523	0.446	52.8	22916	860	26.6	64	1.36	0.021
Expand11	5335	28.462	0.502	56.7	30809	962	32.0	65	1.64	0.025
Expand12	5820	33.872	0.559	60.6	34774	1100	31.6	66	1.51	0.023
Expand13	6305	39.753	0.614	64.7	42243	1220	34.6	73	1.88	0.026
Expand14	6790	46.100	0.674	68.4	51643	1400	36.9	76	1.38	0.018
Expand15	7275	52.926	0.732	72.3	62746	1549	40.5	78	1.20	0.015

Table 1: Results on various problems. In these experiments, k (the number of senior airports) is 3, ϵ (the stopping criterion of Equation 6) is 0.05 and p_{rand} (the probability of the requested move in the maze being replaced by a random move) is 0.1. The Small maze is shown in Figure 1. Medium, Big, OneWay and Expand5 are all shown in Figure 7.

this ability to access $J^\star(x, y)$ and $\pi^\star(x, y)$ useful? Certainly in domains in which we must perform multiple sequential tasks, and in which we need to very quickly switch from task to task. But we believe the greatest use will come in hierarchical control systems in which a higher level controller wishes to not merely call one of a set of atomic lower level controllers, but instead one of a set of lower level controllers that can be parameterized with a subgoal. The full paper and presentation of this work includes an example of hierarchical control of hunters and prey which would not have been otherwise computationally feasible.

Multi-values are primitive in comparison with Parr's approach [Parr and Russell, 1998], which permits arbitrary reward functions instead of merely arbitrary goals. However, multi-value functions may be more computationally practical on large problems. Like other hierarchical RL algorithms, Airports uses abstract actions, and can be thought of as creating an entire plan, and then replanning on each time step. But Airports differs from other algorithms such as MAXQ [Dietterich, 1998] in the nature of those abstract actions, and this can be seen by looking at the plans produced. In MAXQ, the problem is solved by a sequence of level-1 actions. Each level-1 action is, in turn, a sequence of level-2 actions, and so on down to the level of primitive actions. The cost of a level-1 action is unknown until the lower levels have computed. In Airports, the plan at a given time might be a single level-3 action ("go to this level-3 airport"), then a single level-7 action, then a single level-10 action. The sequence is constrained to always be in increasing numerical order (decreasing levels of abstraction), with at most one action at each level, guaranteeing short plans. Each abstract action is composed directly of primitive actions, not lower-level abstract actions, and the complete hierarchy of abstract actions—what they do and where they are applicable—is constructed automatically and parsimoniously. This restriction means that a good plan to get from state x to state y can be found by exhaustively searching *all* legal plans, and this is fast enough to be done in real time on every step. It might be interesting to combine airports with MAXQ or with Precup's and Sutton's abstract actions [Precup and Sutton, 1998], using the abstract actions generated by Airports as some of the "primitive" actions in the other algorithm.

References

[Bertsekas and Tsitsiklis, 1989] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation.* Prentice Hall, 1989.

[Chapman and Kaelbling, 1991] D. Chapman and L. P. Kaelbling. Learning from Delayed Reinforcement In a Complex Domain. In *IJCAI-91*, 1991.

[Dayan and Hinton, 1993] P. Dayan and G. E. Hinton. Feudal Reinforcement Learning. In S. J. Hanson, J. D Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5.* Morgan Kaufmann, 1993.

[Dietterich, 1998] T. G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. In Jude Shavlik, editor, *International Conference on Machine Learning*, 1998.

[Kaelbling, 1993] L. Kaelbling. Hierarchicial Learning in Stochastic Domains: Preliminary Results. In *Machine Learning: Proceedings of the Tenth International Workshop.* Morgan Kaufmann, June 1993.

[Kearns *et al.*, 1998] M. Kearns, Y. Mansour, and A. Ng. Sparse Sampling Methods for Planning and Learning in Large and Partially Observable Markov Decision Processes. Draft Report, 1998.

[Moore and Atkeson, 1993] A. W. Moore and C. G. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning*, 13, 1993.

[Moore, 1994] A. W. Moore. The Parti-game Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State-spaces. In S. J. Hanson, J. D Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 6.* Morgan Kaufmann, April 1994.

[Parr and Russell, 1998] R. Parr and S. Russell. Reinforcement Learning with Hierarchies of Machines. In *Neural Information Processing Systems 10, 1997.* Morgan Kaufmann, 1998.

[Precup and Sutton, 1998] D. Precup and R. Sutton. Multi-Time Models for Temporally Abstract Planning. In *Neural Information Processing Systems 10, 1997.* Morgan Kaufmann, 1998.

[Singh, 1991] S. P. Singh. Transfer of learning across compositions of sequential tasks. In L. Birnbaum and G. Collins, editors, *Machine Learning: Proceedings of the Eighth International Workshop.* Morgan Kaufmann, June 1991.